Honda Research Institute EU

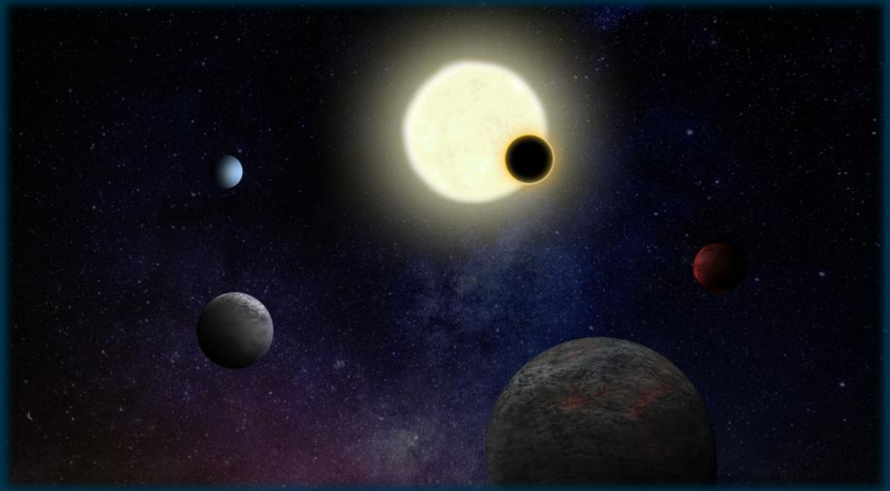
SPOC: Approaches by HRI

Steffen Limmer, Felix Lanfermann, **Nils Einecke**

(disclaimer: we have no background in astrophysics)

Space Optimisation Competition (SpOC) Workshop
16.09.2022

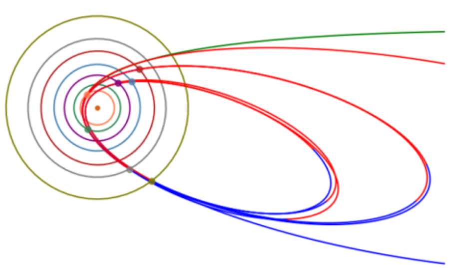




Trappist Tour

SPOC: Trappist Tour Algorithm Overview

```
PS = {} // Pareto set
for UB_T in {50, 75, 100, 125, 150, 175} do
    for iter in 1, ..., ITER do
        planets = random_permutation(7);
        tour = []; // start with an empty tour
        for leg in 1, ..., 6 do
            if leg == 1 then
                // Construct legs from initial point over first planet to
                   second planet
                Set lower bound of T[0] to 1400;
                Set upper bound of T[0] to 1600;
                Set upper bound of T[1] to UB_T;
                tour = tour + optimize_leg(init_point, planets[0], planets[1]);
            else
                // Construct next leg
                Set upper bound of T[leg] to UB_T;
                tour = tour + optimize_leg(planets[leg − 1], planets[leg]);
            end
        end
        tour = local_opt(tour);
        if nondominated(tour) then
            PS = PS + {tour};
        end
    end
end
return PS;
```



- Generative approach, which constructs tour leg by leg
- Randomly drawn sequences of planets
- Single-objective evolutionary algorithm used for optimization of legs
- Optimization of resulting complete tours with CMA-ES
- Vary upper bounds for leg times to tackle multi-objective character of problem
  {50, 75, 100, 125, 150, 175}
- Restrict time for leg from initial point to first planet to interval [1400, 1600]

SPOC: Trappist Tour
Evolutionary Algorithm

Individual encoding:

- Sequence of continuous variables belonging to current leg

Parent selection:

- Binary tournament selection

Crossover:

- Uniform crossover with crossover probability *pc*

Mutation:

- Mutate each gene with mutation operator from breeder genetic algorithm [1] with probability *pm*

Fitness:

- Hypervolume resulting from mission time and fuel consumption of subtour from initial point to currently considered leg
- High penalty for infeasible solutions

Optimization process:

- Steady-state generational scheme with 2 offspring per generation
- If offspring is produced through copying a parent, its mutation is repeated until at least one gene is mutated
- Offspring replaces best individual in population with similar leg time (i.e., the leg time is within that of the offspring +/- 1 day) if there is such an individual and the offspring is not worse than that individual
- Offspring replaces worst individual in population if there is no individual with similar leg time in population and the offspring is better than the worst individual

Implementation:

- Own implementation in Python

[1] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization," in *Evolutionary Computation*, vol. 1, no. 1, pp. 25-49, March 1993.

SPOC: Trappist Tour
CMA-ES

Individual encoding:

- Sequence of continuous variables of a tour
- Normalized to interval [0,1]

Initial solution:

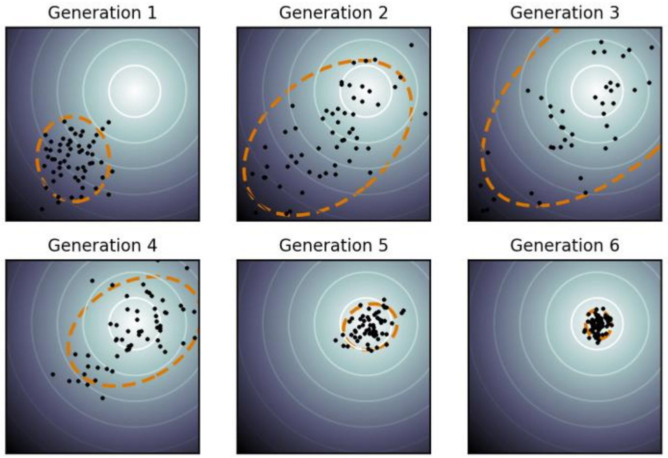
- Parameters of tour resulting from EA

Fitness:

- Fuel consumption plus high penalty for infeasible solutions

Implementation:

- Python module *cma*



SPOC: Trappist Tour
Parameter Setting and Final Result

Overall algorithm:

- ITER=100 iterations per upper bound of leg time

Evolutionary algorithm:

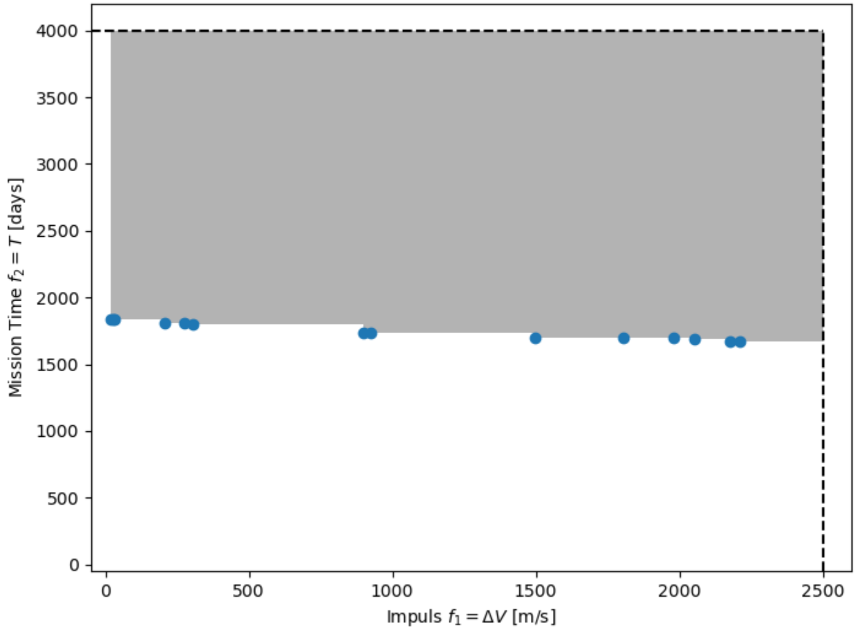
- Population size of 100
- 100,000 generations
- Crossover rate *pc*=0.2
- Mutation rate *pm*=0.1

CMA-ES:

- Initial mutation step size sigma: 0.05
- Population size: Default (4+ ⌊3 x ln(N)⌋)

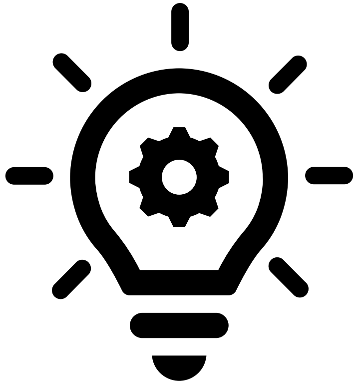
Final result:

- Hypervolume: **5,601,396**
- Number of solutions: 12

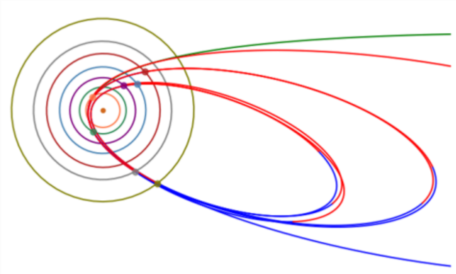


SPOC: Trappist Tour
Approach Observations and Issues

- The **EA is not robust** – large variation in results dependent on seed
- The **EA returns only the best individual** of the final population and the rest of the population is discarded
  - ➢ Potential improvement by considering other good individuals of the final population
- **CMA-ES** can typically find notable **improvements in fuel consumption** but not in mission time
- **CMA-ES** was only beneficial for improving solutions of the EA but **not for optimizing tours or legs from scratch** since it converged too fast to a local optimum
  - We tried different variants of the settings of CMA-ES but were not able to make the optimization more global



SPOC: Trappist Tour
General Insights



One that governs almost everything:

- Major share from one solution ~ 5.4Mio HV
- Other only minor contribution

Only T[0]~1500days leads to almost no fuel (first leg time)

- Probably time it takes with initial speed to arrive at solar system, changing initial velocity takes fuel

Once time is low its easy to keep:

- Finding good start is hard part
- Once time per leg has reached a low value its often rather easy to find a solution in that time range for the next planet

rp is always minimal (1.1):

- We had no major solution which had a larger rp
  → laymen guess: you can steal more momentum the closer you are to a planet, more momentum steal means less fuel

Start with planet 0 is enough

- All found solutions use start planet 0 or 1
- It seems start planet 0 would also be enough
- We have no laymen explanation for this

SPOC: Trappist Tour
Algorithm Overview (Alternative Approach)

EA variation

- Draw 100 random tours (starting with planet 0)

- Optimize each tour for fuel consumption leg by leg
  - Entry time into system fixed to 1500 days
  - NSGA2 from pymoo with DV fitness and varying maximal leg times
    - Repeat NSGA2 several times (for each leg) to tackle local minima problem
    - Note: good results with fixed maximal leg times [135, 50, 50, 50, 35, 35]
  - NSGA2 setting default, except n_gen = 100

- Compute fitness for each tour and remove dominated solutions

- Result:
  - Essentially same result as with other approach

pymoo
Multi-objective Optimization in Python

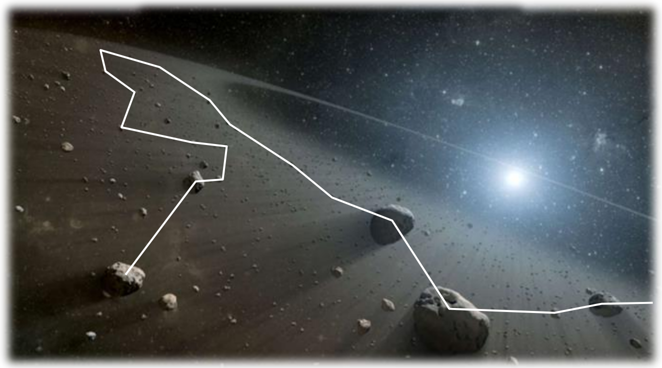
THANK YOU FOR YOUR ATTENTION

“So much universe, so little time.”

Terry Pratchett

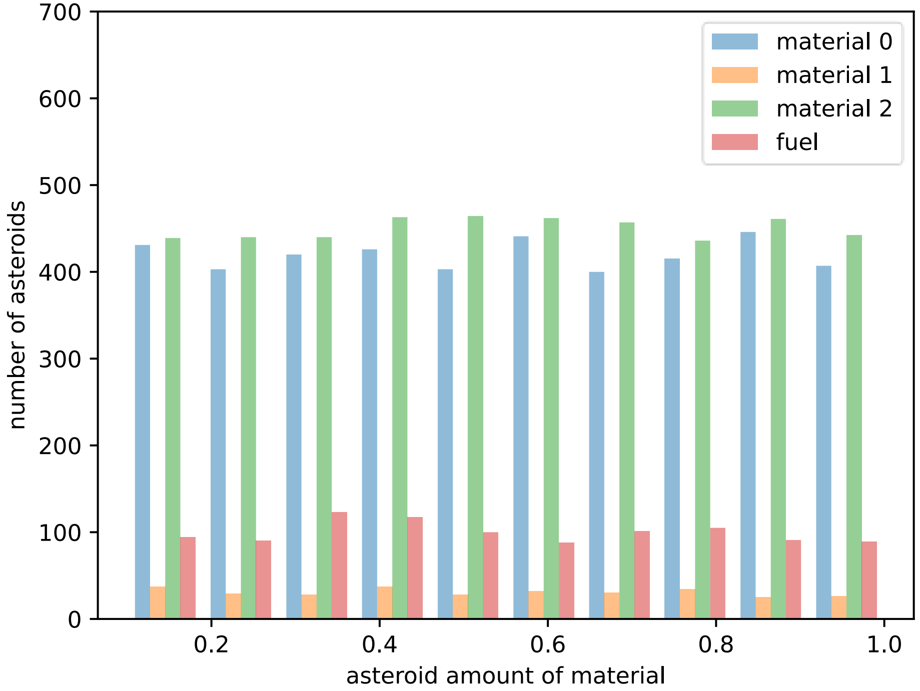
Mining

SPOC: Mining Algorithm Overview



Greedy approach

- Iterate over all asteroids using them as start asteroids
  - *Iterate until fuel or days consumed*
    - Find best next asteroid for each material type [0, 1, 2] where collected material is <= γ
    - Go to best asteroid except cost for material-1-asteroid is below ρ → material 1 is scares → take good opportunities
    - If all material > γ then search only for least collected material type → balance materials after some time
    - If fuel falls below 0.2 → got to best fuel asteroid
  - *Endgame-optimization*
    - Do not try to fly to fuel asteroid later than day 1750
    - If time left, try to fly to any asteroid of material of interest (ignoring tof and fuel threshold)
    - Mine last asteroid also beyond mission time



SPOC: Mining
Cost to Mine Material

Idea

- The main resource/cost here is time
- Fuel can be transformed into time
  - Short flight times cost more fuel
  - It takes time to refuel (go to fuel asteroid and mine fuel)
- Getting a material should be as cheap as possible
  - Cost (time used) per material gathered
- Cost elements
  - Fly to asteroid: time_of_flight
  - Mine the asteroid: time_to_mine
  - Time cost caused by fuel usage: time_to_refuel * fuel_used
- Normalize cost
  - Divide cost by amount of material gathered

Definition of cost:

$$mC(a, tof) = \frac{\boxed{tof + tm(a) + ft * fc(tof)}}{\boxed{m(a)}}$$

Legend:

mC	cost for gathering material [day/material]
tof	time of flight [day]
tm(a)	time to fully mine asteroid a [day]
m(a)	mass of material of asteroid a [material]
ft	average time to fully fuel tank (set to 80) [day]
fc	fuel consumption of flight [fuel]

SPOC: Mining Cost to Refuel

Idea

- Also, for refueling actual cost is time

- Cost elements
  - Fly to fuel asteroid: time_of_flight
  - Mine the asteroid: time_to_mine

- Normalize cost
  - Divide cost by amount of fuel refueled
  - Note1: refuel is "missing in tank" + "fuel used to reach refuel asteroid"
  - Note2: asteroid may not have enough fuel to fill up completely

Definition of cost:

$$fC(a, tof) = \frac{\boxed{tof + 30 * fh(a, tof)}}{\boxed{fh - fc}}$$

$$fh(a, tof) = min[m(a),\ (1.0 - sof) + fc(tof)]$$

Legend:

fC	cost for gathering fuel [day/fuel]
tof	time of flight [day]
30	time for harvesting 1.0 fuel [day]
m(a)	fuel available at asteroid a [fuel]
fc	fuel consumption of flight [fuel]
fh	fuel harvested [fuel]
sof	state of fuel (0.0, 1.0) [fuel]

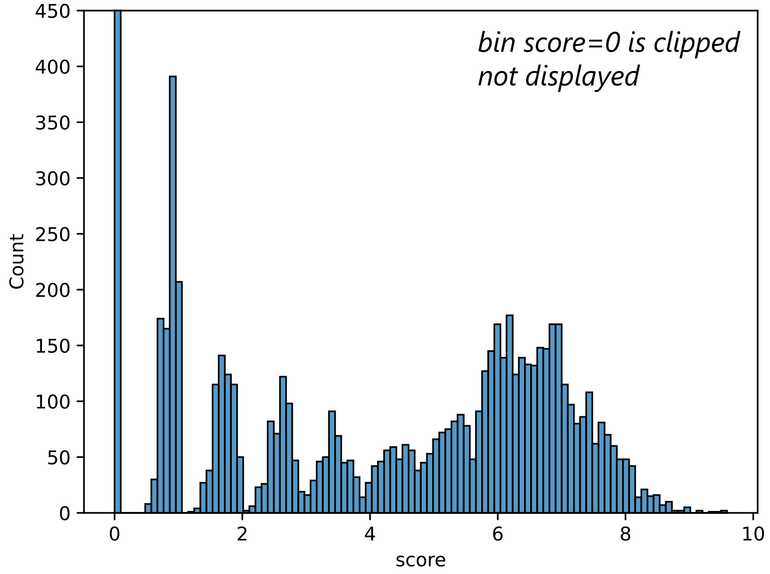
SPOC: Mining
Parameter Setting and Final Result

Setting:

- γ=9.5
- ρ=80

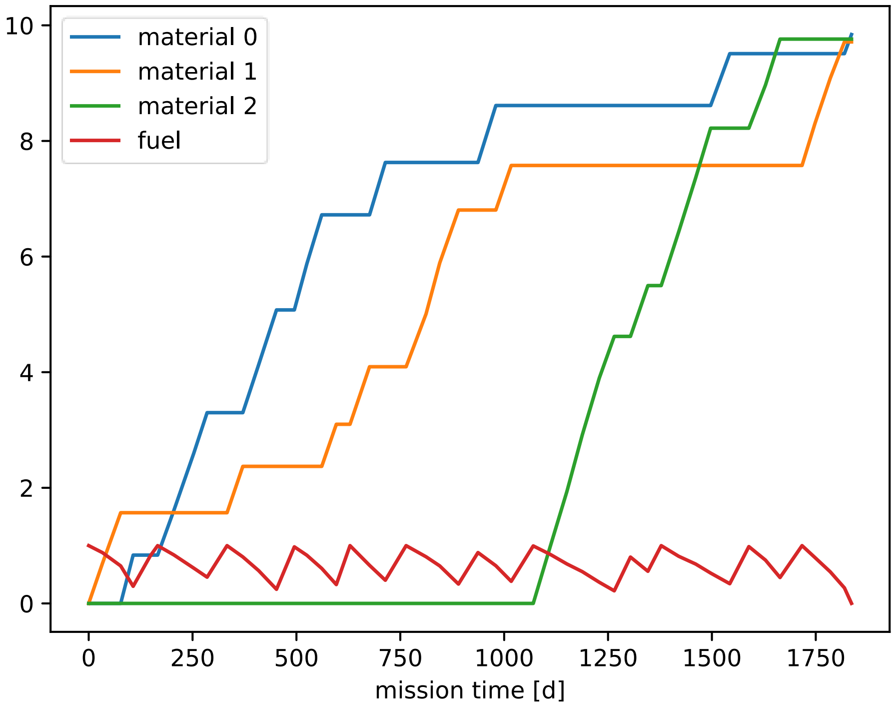
- maximal considered tof
  [m0, m1, m2, fuel] → [20, 25, 20, 35]

- consider only asteroids with enough material
  [m0, m1, m2, fuel] → [0.7, 0.5, 0.7, 0.5]



Final Result:

- **Score = 9.713**
  m0 = 9.841 | m1 = 9.713 | m2 = 9.762



SPOC: Mining Insights

Defined cost has always a clear global minimum:

- Restriction of tof mainly to reduce compute time
- Potential speed improvement → stop iterating over tof when cost rises again

Material 1 is scarce:

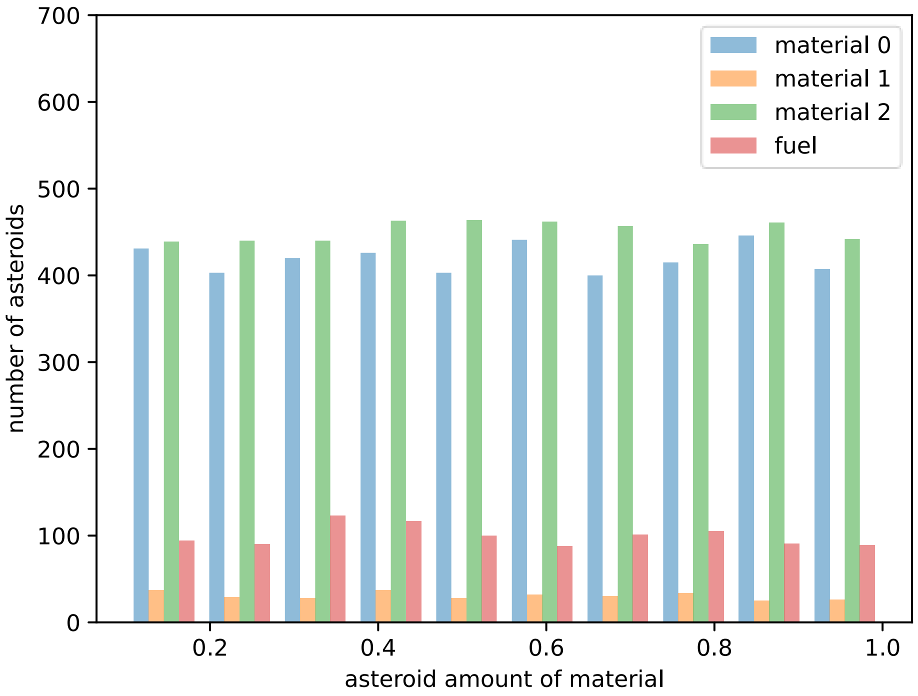
- Thus, we decided to prioritize during search if good enough opportunity

Fuel is sometimes tricky:

- Often fuel is unreachable or very costly
  → Room for improvement (i.e. backtracking or so)

We did not take asteroid data into account:

- Room for improvement?
- Maybe to preselect potential start asteroids?



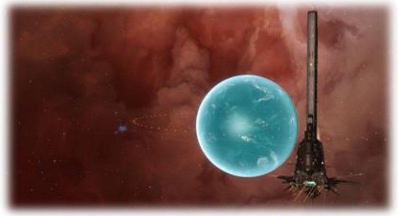
THANK YOU FOR YOUR ATTENTION

"So much universe, so little time."

Terry Pratchett

Delivery Schedule

SPOC: Delivery Scheduling Algorithm Overview



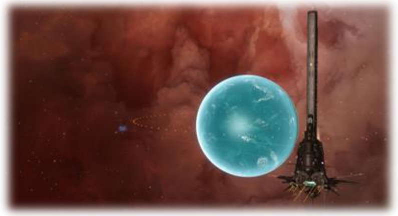
Bilevel approach

- Outer Level:
  - Optimization of time window ends (11 continuous variables) and order of time windows (12 integer variables)
  - Hand-tailored ***evolutionary algorithm***

- Inner Level:
  - Optimization of deliveries for fixed time windows
  - Formulated and solved as mixed integer linear programming problem (***MILP***)
  - Problem solved with Gurobi solver
  - Optimality **gap** tolerance set to **1%** to speed-up optimization

- Fine tuning by applying ***MILP*** with result as start solution with optimality **gap** of **0.01%**

SPOC: Delivery Scheduling Evolutionary Algorithm



Individual encoding:

- 2 sequences: One sequence of (sorted) continuous variables for time window ends and one sequence of integer variables encoding assignment of stations to time windows

Parent selection:

- Binary tournament selection

Crossover:

- Station assignments are just copied from parents
- Blend (BLX) crossover for continuous variables ([p1-alpha … p2+alpha])

Mutation:

- Each integer variable is swapped with another random integer variable with probability *ps*
- Each continuous variable is mutated with Gaussian mutation with probability *pg (per gene)*

Fitness:

- MILP is used to optimize the deliveries for the time windows and stations assignments encoded in the individual

Optimization process:

- Steady-state generational scheme with 2 offspring per generation
- If offspring is produced through copying a parent, its mutation is repeated until at least one gene is mutated
- Offspring replaces best individual in population with similar station assignment if there is such an individual and the offspring is not worse than that individual
- Offspring replaces worst individual in population if there is no individual with similar station assignment in population and the offspring is better than the worst individual

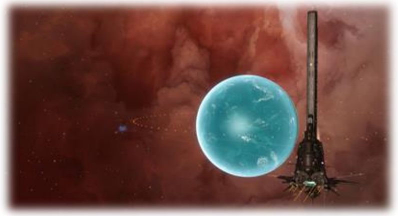
Handling of Constraints:

- Initialization, crossover and mutation of window ends is repeated until resulting window ends are feasible

Implementation:

- Own implementation in Python

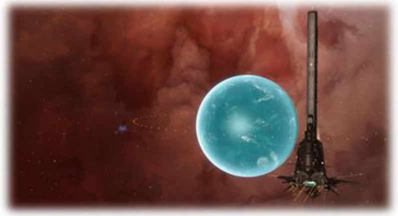
SPOC: Delivery Scheduling
MILP Approach and Improvement of Results



- Gurobi solver version 9.1 is used for MILP optimizations
- Optimality gap tolerance is set to 1% in fitness evaluation to speed up MILP optimization
- Final result is improved in two ways:
  - Optimization of deliveries with MILP with default optimality gap tolerance of 0.01%
  - The complete optimization problem is formulated and solved as MILP problem with the currently best result as initial solution

GUROBI OPTIMIZATION

SPOC: Trappist Tour
Parameter Setting and Final Results



Evolutionary algorithm:

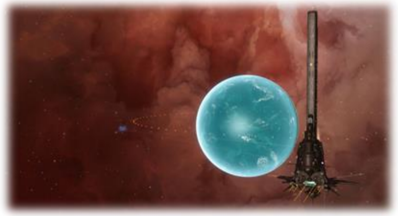
- Population size of 100
- 100,000 generations
- Crossover rate *pc*=0.5
- Alpha parameter of BLX crossover alpha=0.4
- Probability for swap of a station assignment in mutation *ps*=0.05
- Probability for Gaussian mutation of a time window end *pg*=0.1

Final Results:

- Result of hybrid EA:
  - 9.484647
- Improved result after applying MILP with gap of 0.01% on solution with time limit of 10 minutes:
  - 9.51402
- Further improved result after applying final MILP for complete problem on solution with time limit of 8 hours:
  - **9.51885**
- Further improved result after applying final MILP for complete problem on solution again with time limit of 12 hours:
  - 9.52579*

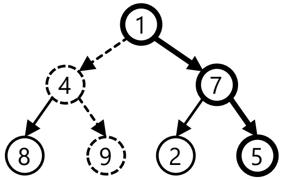
* submitted post challenge

SPOC: Delivery Scheduling Algorithm Overview (Alternative Approach)



Random-Greedy approach

- Iterate a few hundred times
  - Draw windows randomly (draw from asteroid arrival times)
  - Iterate ~10 times
    - Randomly assign station to windows
    - Greedy assignment of each asteroid to station
      → where yielding best min material



-
  -
    - Equalize station
      → move asteroid with best min material to worst station
      → iterate until no further improvement found
    - Greedy switch station (check if switch of time windows improves result)

- Results around 9.0-9.1
  → best (outlier) result 9.26

THANK YOU FOR YOUR ATTENTION

"So much universe, so little time."

Terry Pratchett